

25 | 特征工程（下）：有哪些常用的特征处理函数？

2021-11-08 吴磊

《零基础入门Spark》

课程介绍 >



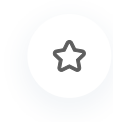
讲述：吴磊

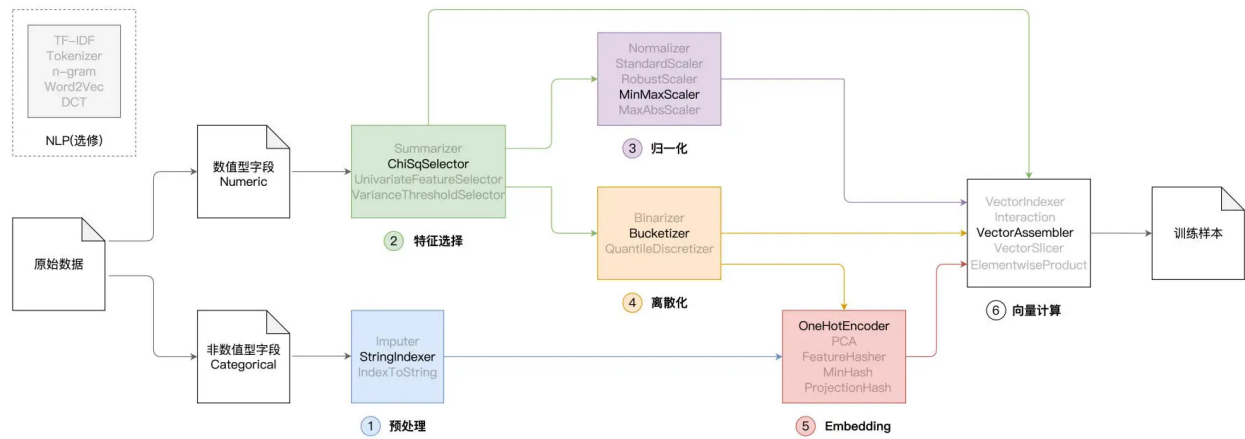
时长 15:57 大小 14.62M



你好，我是吴磊。

在上一讲，我们提到，典型的特征工程包含如下几个环节，即预处理、特征选择、归一化、离散化、Embedding 和向量计算，如下图所示。



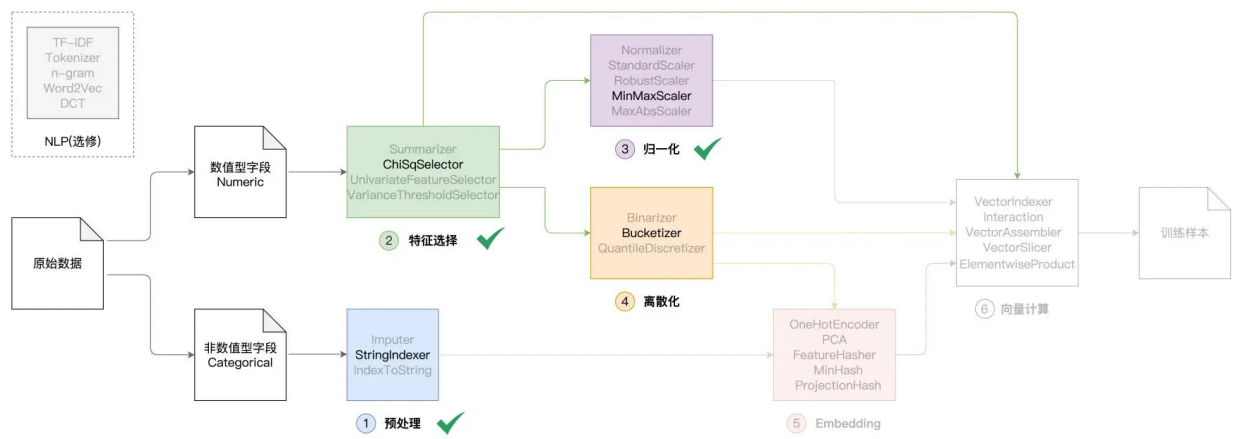


特征工程一览 & Spark MLlib特征处理函数分类

在上一讲，我们着重讲解了其中的前 3 个环节，也就是预处理、特征选择和归一化。按照之前的课程安排，今天这一讲，咱们继续来说说剩下的离散化、Embedding 与向量计算。

特征工程是机器学习的重中之重，只要你耐心学下去，必然会不虚此行。这一讲的最后，我还会对应用了 6 种不同特征工程的模型性能加以对比，帮你深入理解特征工程中不同环节的作用与效果。

特征工程



特征工程进度打卡

在上一讲，我们打卡到了“第三关”：归一化。因此，接下来，我们先从“第四关”：离散化说起。

离散化：Bucketizer

与归一化一样，离散化也是用来处理数值型字段的。离散化可以把原本连续的数值打散，从而降低原始数据的多样性（Cardinality）。举例来说，“BedroomAbvGr”字段的含义是居室数量，在 train.csv 这份数据样本中，“BedroomAbvGr”包含从 1 到 8 的连续整数。

现在，我们根据居室数量，把房屋粗略地划分为小户型、中户型和大户型。



BedroomAbvGr离散化

不难发现，“BedroomAbvGr”离散化之后，数据多样性由原来的 8 降低为现在的 3。那么问题来了，原始连续数据好好的，为什么要对它做离散化呢？**离散化的动机，主要在于提升特征数据的区分度与内聚性，从而与预测标的产生更强的关联。**

就拿“BedroomAbvGr”来说，我们认为一居室和两居室对于房价的影响差别不大，同样，三居室和四居室之间对于房价的影响，也是微乎其微。

但是，小户型与中户型之间，以及中户型与大户型之间，房价往往会出现跃迁的现象。换句话说，相比居室数量，户型的差异对于房价的影响更大、区分度更高。因此，把“BedroomAbvGr”做离散化处理，目的在于提升它与预测标的之间的关联性。

那么，在 Spark MLlib 的框架下，离散化具体该怎么做呢？与其他环节一样，Spark MLlib 提供了多个离散化函数，比如 Binarizer、Bucketizer 和 QuantileDiscretizer。我们不妨以 Bucketizer 为代表，结合居室数量 “BedroomAbvGr” 这个字段，来演示离散化的具体用法。老规矩，还是先上代码为敬。

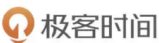
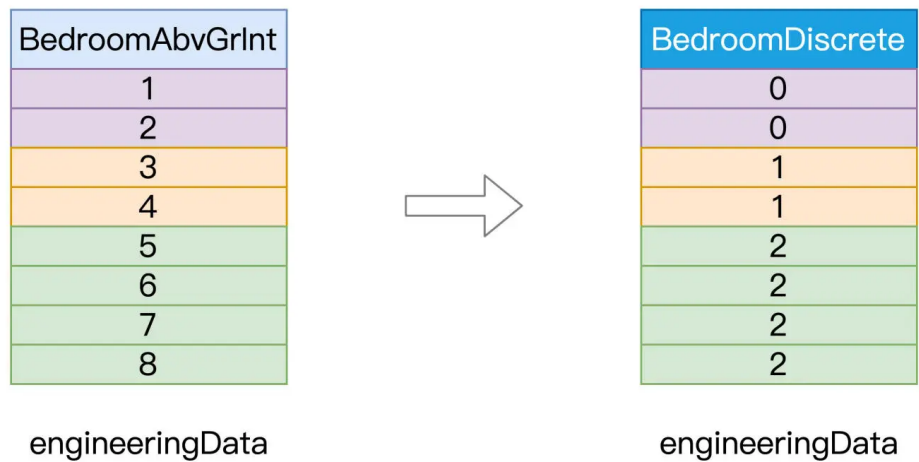
 复制代码

```
1 // 原始字段
2 val fieldBedroom: String = "BedroomAbvGrInt"
3 // 包含离散化数据的目标字段
4 val fieldBedroomDiscrete: String = "BedroomDiscrete"
5 // 指定离散区间，分别是[负无穷, 2]、[3, 4]和[5, 正无穷]
6 val splits: Array[Double] = Array(Double.NegativeInfinity, 3, 5, Double.PositiveInfinity)
7
8 import org.apache.spark.ml.feature.Bucketizer
9
10 // 定义并初始化Bucketizer
11 val bucketizer = new Bucketizer()
12 // 指定原始列
13 .setInputCol(fieldBedroom)
14 // 指定目标列
15 .setOutputCol(fieldBedroomDiscrete)
16 // 指定离散区间
17 .setSplits(splits)
18
19 // 调用transform完成离散化转换
20 engineeringData = bucketizer.transform(engineeringData)
```

不难发现，Spark MLlib 提供的特征处理函数，在用法上大同小异。首先，我们创建 Bucketizer 实例，然后将数值型字段 BedroomAbvGrInt 作为参数传入 setInputCol，同时使用 setOutputCol 来指定用于保存离散数据的新字段 BedroomDiscrete。

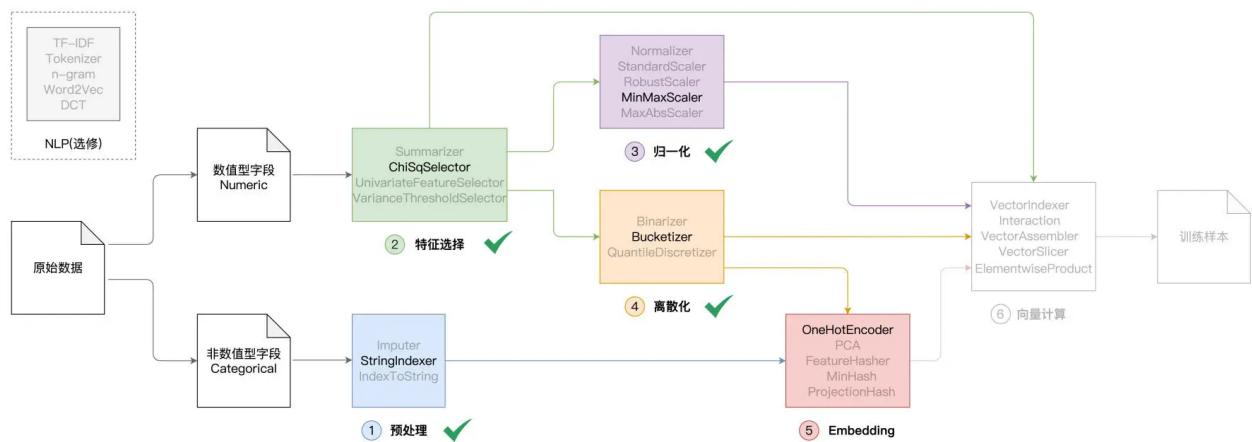
离散化的过程是把连续值打散为离散值，但具体的离散区间如何划分，还需要我们通过在 setSplits 里指定。离散区间由浮点型数组 splits 提供，从负无穷到正无穷划分出了[负无穷, 2]、[3, 4]和[5, 正无穷]这三个区间。最终，我们调用 Bucketizer 的 transform 函数，对 engineeringData 做离散化。

离散化前后的数据对比，如下图所示。



离散化前后数据对比

好啦，到此为止，我们以 Bucketizer 为代表，学习了 Spark MLlib 框架中数据离散化的用法，轻松打通了特征工程的第四关。



打卡特征工程第四关：离散化

Embedding

实际上，Embedding 是一个非常大的话题，随着机器学习与人工智能的发展，Embedding 的方法也是日新月异、层出不穷。从最基本的热独编码到 PCA 降维，从 Word2Vec 到 Item2Vec，从矩阵分解到基于深度学习的协同过滤，可谓百花齐放、百家

争鸣。更有学者提出：“万物皆可 Embedding”。那么问题来了，什么是 Embedding 呢？

Embedding 是个英文术语，如果非要找一个中文翻译对照的话，我觉得“向量化”（Vectorize）最合适。Embedding 的过程，就是把数据集合映射到向量空间，进而把数据进行向量化的过程。这句话听上去有些玄乎，我换个更好懂的说法，Embedding 的目标，就是找到一组合适的向量，来刻画现有的数据集合。

以 GarageType 字段为例，它有 6 个取值，也就是说我们总共有 6 种车库类型。那么对于这 6 个字符串来说，我们该如何用数字化的方式来表示它们呢？毕竟，模型只能消费数值，不能直接消费字符串。

Attchd	Detchd	CarPort	BuiltIn	Basment	2Types
--------	--------	---------	---------	---------	--------

 极客时间

GarageType 的 6 个取值

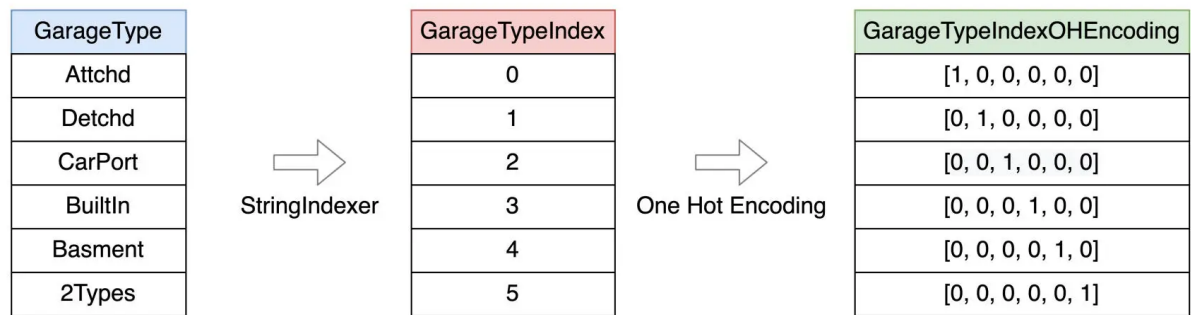
一种方法是采用预处理环节的 StringIndexer，把字符串转换为连续的整数，然后让模型去消费这些整数。在理论上，这么做没有任何问题。但从模型的效果出发，整数的表达方式并不合理。为什么这么说呢？

我们知道，连续整数之间，是存在比较关系的，比如 $1 < 3$ ， $6 > 5$ ，等等。但是原始的字符串之间，比如，“Attchd”与“Detchd”并不存在大小关系，如果强行用 0 表示“Attchd”、用 1 表示“Detchd”，逻辑上就会出现“Attchd” $<$ “Detchd”的悖论。

因此，预处理环节的 StringIndexer，仅仅是把字符串转换为数字，转换得到的数值是不能直接喂给模型做训练。我们需要把这些数字进一步向量化，才能交给模型去消费。那么问

题来了，对于 StringIndexer 输出的数值，我们该怎么对他们进行向量化呢？这就要用到 Embedding 了。

作为入门课，咱们不妨从最简单的**热独编码（One Hot Encoding）**开始，去认识 Embedding 并掌握它的基本用法。我们先来说说，热独编码，是怎么一回事。相比照本宣科说概念，咱们不妨以 GarageType 为例，从示例入手，你反而更容易心领神会。



热独编码示例

首先，通过 StringIndexer，我们把 GarageType 的 6 个取值分别映射为 0 到 5 的六个数值。接下来，使用热独编码，我们把每一个数值都转化为一个向量。

向量的维度为 6，与原始字段（GarageType）的多样性（Cardinality）保持一致。换句话说，热独编码的向量维度，就是原始字段的取值个数。

仔细观察上图的六个向量，只有一个维度取值为 1，其他维度全部为 0。取值为 1 的维度与 StringIndexer 输出的索引相一致。举例来说，字符串 “Attchd” 被 StringIndexer 映射为 0，对应的热独向量是 [1, 0, 0, 0, 0, 0]。向量中索引为 0 的维度取值为 1，其他维度全部取 0。

不难发现，热独编码是一种简单直接的 Embedding 方法，甚至可以说是 “简单粗暴”。不过，在日常的机器学习开发中，“简单粗暴” 的热独编码却颇受欢迎。

接下来，我们还是从“房价预测”的项目出发，说一说热独编码的具体用法。

在预处理环节，我们已经用 `StringIndexer` 把非数值字段全部转换为索引字段，接下来，我们再用 `OneHotEncoder`，把索引字段进一步转换为向量字段。

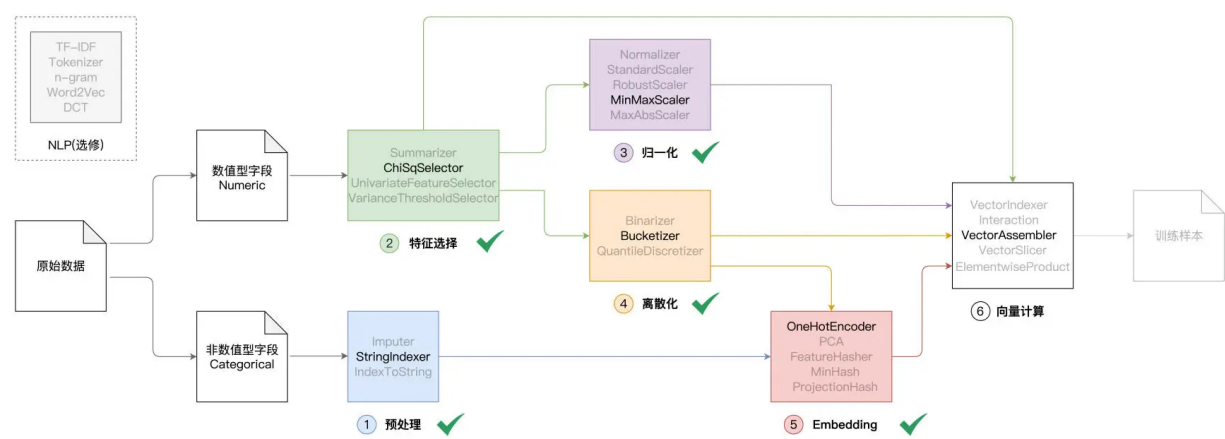
 复制代码

```
1 import org.apache.spark.ml.feature.OneHotEncoder
2
3 // 非数值字段对应的目标索引字段，也即StringIndexer所需的“输出列”
4 // val indexFields: Array[String] = categoricalFields.map(_ + "Index").toArray
5
6 // 热独编码的目标字段，也即OneHotEncoder所需的“输出列”
7 val oheFields: Array[String] = categoricalFields.map(_ + "OHE").toArray
8
9 // 循环遍历所有索引字段，对其进行热独编码
10 for ((indexField, oheField) <- indexFields.zip(oheFields)) {
11   val oheEncoder = new OneHotEncoder()
12   .setInputCol(indexField)
13   .setOutputCol(oheField)
14   engineeringData = oheEncoder.transform(engineeringData)
15 }
```

可以看到，我们循环遍历所有非数值特征，依次创建 `OneHotEncoder` 实例。在实例初始化的过程中，我们把索引字段传入给 `setInputCol` 函数，把热独编码目标字段传递给 `setOutputCol` 函数。最终通过调用 `OneHotEncoder` 的 `transform`，在 `engineeringData` 之上完成转换。

好啦，到此为止，我们以 `OneHotEncoder` 为代表，学习了 Spark MLlib 框架中 `Embedding` 的用法，初步打通了特征工程的第五关。

尽管还有很多其他 `Embedding` 方法需要我们进一步探索，不过从入门的角度来说，`OneHotEncoder` 完全可以应对大部分机器学习应用。



打卡特征工程第五关：Embedding

向量计算

打通第五关之后，特征工程“这套游戏”还剩下最后一道关卡：向量计算。

向量计算，作为特征工程的最后一个环节，主要用于构建训练样本中的特征向量（Feature Vectors）。在 Spark MLlib 框架下，训练样本由两部分构成，第一部分是预测标的（Label），在“房价预测”的项目中，Label 是房价。

而第二部分，就是特征向量，在形式上，特征向量可以看作是元素类型为 Double 的数组。根据前面的特征工程流程图，我们不难发现，特征向量的构成来源多种多样，比如原始的数值字段、归一化或是离散化之后的数值字段、以及向量化之后的特征字段，等等。

Spark MLlib 在向量计算方面提供了丰富的支持，比如前面介绍过的、用于集成特征向量的 VectorAssembler，用于对向量做剪裁的 VectorSlicer，以元素为单位做乘法的 ElementwiseProduct，等等。灵活地运用这些函数，我们可以随意地组装特征向量，从而构建模型所需的训练样本。

在前面的几个环节中（预处理、特征选择、归一化、离散化、Embedding），我们尝试对数值和非数值类型特征做各式各样的转换，目的在于探索可能对预测标的影响更大的潜在因素。

接下来，我们使用 VectorAssembler 将这些潜在因素全部拼接在一起、构建特征向量，从而为后续的模式训练准备好训练样本。

 复制代码

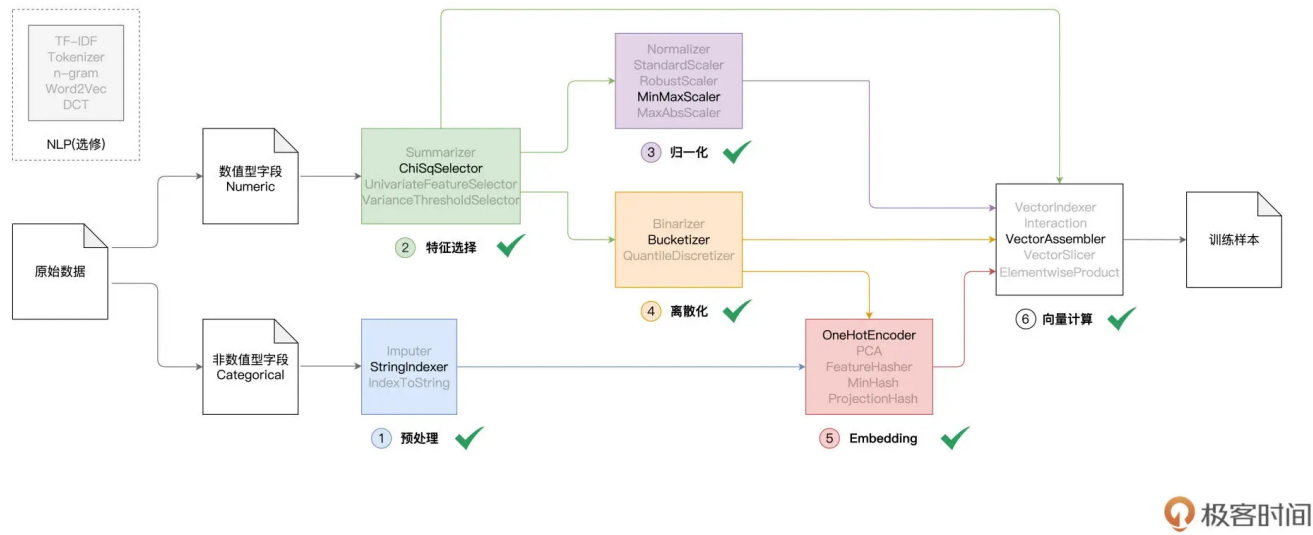
```
1 import org.apache.spark.ml.feature.VectorAssembler
2
3 /**
4  入选的数值特征：selectedFeatures
5  归一化的数值特征：scaledFields
6  离散化的数值特征：fieldBedroomDiscrete
7  热独编码的非数值特征：oheFields
8  */
9
10 val assembler = new VectorAssembler()
11 .setInputCols(selectedFeatures ++ scaledFields ++ fieldBedroomDiscrete ++ oheF
12 .setOutputCol("features")
13
14 engineeringData = assembler.transform(engineeringData)
```

转换完成之后，`engineeringData` 这个 `DataFrame` 就包含了一列名为 “features” 的新字段，这个字段的内容，就是每条训练样本的特征向量。接下来，我们就可以像上一讲那样，通过 `setFeaturesCol` 和 `setLabelCol` 来指定特征向量与预测标的，定义出线性回归模型。

 复制代码

```
1 // 定义线性回归模型
2 val lr = new LinearRegression()
3 .setFeaturesCol("features")
4 .setLabelCol("SalePriceInt")
5 .setMaxIter(100)
6
7 // 训练模型
8 val lrModel = lr.fit(engineeringData)
9
10 // 获取训练状态
11 val trainingSummary = lrModel.summary
12 // 获取训练集之上的预测误差
13 println(s"Root Mean Squared Error (RMSE) on train data: ${trainingSummary.root
```

好啦，到此为止，我们打通了特征工程所有关卡，恭喜你！尽管不少关卡还有待我们进一步去深入探索，但这并不影响我们从整体上把握特征工程，构建结构化的知识体系。对于没讲到的函数与技巧，你完全可以利用自己的碎片时间，借鉴这两节课我给你梳理的学习思路，来慢慢地将它们补齐，加油！



通关！

通关奖励：模型效果对比

学习过 VectorAssembler 的用法之后，你会发现，特征工程任一环节的输出，都可以用来构建特征向量，从而用于模型训练。在介绍特征工程的部分，我们花了大量篇幅，介绍不同环节的作用与用法。

你可能会好奇：“这些不同环节的特征处理，真的会对模型效果有帮助吗？毕竟，折腾了半天，我们还是要看模型效果的”。

没错，特征工程的最终目的，是调优模型效果。接下来，通过将不同环节输出的训练样本喂给模型，我们来对比不同特征处理方法对应的模型效果。

编号	特征工程	训练效果	代码地址
0	上一讲的初始模型	Train RMSE = 48372	调优对比基准
1	选取所有数值字段	Train RMSE = 43562	特征工程-调优1
2	特征选择	Train RMSE = 43250	特征工程-调优2
3	归一化	Train RMSE = 33258	特征工程-调优3
4	离散化	Train RMSE = 41630	特征工程-调优4
5	预处理 + Embedding	Train RMSE = 21522	特征工程-调优5
6	叠加以上所有环节	Train RMSE = 11548	特征工程-调优6

特征工程不同环节优化过后的模型效果

不同环节对应的代码地址如下：

1. [🔗 调优对比基准](#)
2. [🔗 特征工程 - 调优 1](#)
3. [🔗 特征工程 - 调优 2](#)
4. [🔗 特征工程 - 调优 3](#)
5. [🔗 特征工程 - 调优 4](#)
6. [🔗 特征工程 - 调优 5](#)
7. [🔗 特征工程 - 调优 6](#)

可以看到，随着特征工程的推进，模型在训练集上的预测误差越来越小，这说明模型的拟合能力越来越强，而这也意味着，特征工程确实有助于模型性能的提升。

对应特征工程不同环节的训练代码，我整理到了最后的“[🔗 代码地址](#)”那一列。强烈建议你动手运行这些代码，对比不同环节的特征处理方法，以及对应的模型效果。

当然，我们在评估模型效果的时候，不能仅仅关注它的拟合能力，更重要的是模型的泛化能力。拟合能力强，只能说明模型在训练集上的预测误差足够小；而泛化能力，量化的是模型在测试集上的预测误差。换句话说，泛化能力的含义是，模型在一份“未曾谋面”的数据集上表现如何。

这一讲，咱们的重点是特征工程，因此暂时忽略了模型在测试集上的表现。从下一讲的模型训练开始，对于模型效果，我们将同时关注模型这两方面的能力：拟合与泛化。

重点回顾

好啦，今天的内容讲完啦，我们一起来做个总结。今天这一讲，我们主要围绕着特征工程中的离散化、Embedding 和向量计算展开，你需要掌握其中最具代表性的特征处理函数。

到此为止，Spark MLlib 特征工程中涉及的 6 大类特征处理函数，我们就都讲完了。为了让你对他们有一个整体上的把握，同时能够随时回顾不同环节的作用与效果，我把每一个大类的特点、以及咱们讲过的处理函数，都整理到了如下的表格中，供你参考。

序号	关卡名称	关键特点	特征处理函数（我们讲到的）
1	预处理	将模型无法直接消费的数据，转换为可以消费的数据形式	StringIndexer，负责对非数值型特征做初步处理，将字符串转换为数值
2	特征选择	提取与预测标的关联度更高的特征，从而精简模型尺寸、提升模型泛化能力	ChiSqSelector，以卡方检验为基础，选择相关度最高的前N个特征
3	归一化	去掉不同特征之间量纲的影响，避免因量纲不一致而导致的梯度下降震荡、模型收敛效率低下等问题	MinMaxScaler，将任何数据列映射到指定值域，这个值域通常是[0, 1]
4	离散化	降低连续数值特征的多样性，让特征与预测标的之间建立更强的关联性	Bucketizer，把不同数值映射到对应区间，完成连续特征离散化的过程
5	Embedding	Embedding将特征映射到向量空间，以向量形式刻画特征本身	热独编码（OneHotEncoder）
6	向量计算	负责完成向量的拆分、拼接、运算，从而构建特征向量（Feature Vectors），进而生成模型可消费的训练样本	VectorAssembler，将多列拼接为特征向量



今天的内容很多，需要我们多花时间去消化。受 2/8 理论的支配，在机器学习实践中，特征工程往往会花费我们 80% 的时间和精力。由于特征工程制约着模型效果的上限，因此，尽管特征工程的步骤繁多、过程繁琐，但是我们千万不能在这个环节偷懒，一定要认真对待。

这也是为什么我们分为上、下两部分来着重讲解特征工程，从概览到每一个环节，从每一个环节的作用到它包含的具体方法。数据质量构筑了模型效果的天花板，特征工程道阻且长，然而行则将至，让我们一起加油！

每课一练

结合上一讲，对于我们介绍过的所有特征处理函数，如 StringIndexer、ChiSqSelector、MinMaxScaler、Bucketizer、OneHotEncoder 和 VectorAssembler，你能说说他们之间的区别和共同点吗？

欢迎你在留言区记录你的收获与思考，也欢迎你向更多同事、朋友分享今天的内容，说不定就能帮他解决特征工程方面的问题。

分享给需要的人，Ta订阅后你可得 **20 元现金奖励**

 生成海报并分享

 赞 1  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 24 | 特征工程（上）：有哪些常用的特征处理函数？

下一篇 26 | 模型训练（上）：决策树系列算法详解

更多学习推荐

2021 最新大厂 Go 工程师面试真题

大厂面试真题 + 金九银十全新整理 + 核心知识全覆盖

免费领取 



精选留言

 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

